**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования**

**«МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ»**

**(МТУСИ)**

Кафедра «Информационная безопасность»

ЛАБОРАТОРНАЯ РАБОТА №6

по дисциплине:

«Объектно-ориентированное программирование систем защиты информации»

на тему:

«Одиночное и множественное наследование. Виртуальные и чисто виртуальные функции. Абстрактные и конкретные классы»

Выполнила:

студентка 3 курса

группы БПЗ1801

Яковлева К.А.

Вариант №27(12)

Москва 2021

1. Общее задание

Для динамической структуры данных, разработанной в предыдущей лабораторной работе (стек, очередь или дек) создать абстрактный класс, выделив в него необходимые операции.

Унаследовать разработанный в предыдущей лабораторной работе класс от созданного в этой работе абстрактного класса.

Разработать ещё одну реализацию динамической структуры данных, указанной в индивидуальном задании. Разработать соответствующий класс, унаследовав его от абстрактного класса и определив все требуемые операции. Предусмотреть конструкторы инициализации, копирования, перемещения, деструктор, функции вставки и удаления элемента, просмотра доступного элемента и функцию, проверяющую наличие элементов.

Перегрузить операции присваивания, перемещения и потокового вывода для вывода содержимого динамической структуры на экран.

Класс разработать в варианте шаблона.

Создать функцию, получающую указатель на базовый класс и демонстрирующую работу

Создать функцию, получающую ссылку на базовый класс и демонстрирующую работу.

1. Индивидуальное задание

### Таблица 1 – Индивидуальное задание.

|  |  |  |
| --- | --- | --- |
| 12 | Дек на основе однонаправленного списка | StackBasedOnUnidirectionalCyclicLinkedList |

1. Программный код

Листинг 1.1 – Исходный код файла task1.cpp.

#include <iostream>

#include "deque1.cpp"

using namespace std;

int main()

{

int arr, pos, element;

cout << "Deque Test Program" << endl << "Data:" << endl << "Deque Length = ";

cin >> arr;

DequeTemp<int> a(arr);

cout << "Deque First Element = ";

cin >> element;

a.GetFirst(element);

cout << "Deque Fill Element" << endl;

while (a.GetSize() < arr)

{

cout << "[0] - Add element to Front Position" << endl << "[1] - Add element to Backward Position" << endl;

cin >> pos;

if (pos == 0)

{

cout << "Element " << a.GetSize() + 1 << " / " << arr << ": ";

cin >> element;

a.PushFront(element);

}

else if (pos == 1)

{

cout << "Element " << a.GetSize() + 1 << " / " << arr << ": ";

cin >> element;

a.PushBack(element);

}

}

cout << endl << "Deque: " << a << endl;

system("pause");

while (true)

{

cout << "Deque Main Menu" << endl << "[1] Show Deque first element" << endl << "[2] Show Deque last element" << endl << "[3] - Delete Front Deque Element"

<< endl << "[4] - Delete Last Deque Element" << endl << "[5] - Exit" << endl << "Selected Item = ";

cin >> pos;

switch (pos)

{

case 1:

{

cout << a.PeekFront() << endl;

break;

}

case 2:

{

cout << a.PeekBack() << endl;

break;

}

case 3:

{

a.PopFront();

cout << endl << "Deque: " << a << endl;

break;

}

case 4:

{

a.PopBack();

cout << endl << "Deque: " << a << endl;

break;

}

case 5:

{

return 0;

}

default:

{

cout << "Error. Selected Item not exist.";

}

}

}

}

}

case 4:

{

a.PopBack();

cout << endl << "Deque: " << a << endl;

break;

}

case 5:

{

return 0;

}

default:

{

cout << "Error. Selected Item not exist.";

}

}

}

}

Листинг 1.2 – Исходный код файла deque1.cpp.

#include <iostream>

using namespace std;

template<typename T>

class Base

{

public:

virtual void PushFront(T element) {}

virtual void PushBack(T element) {}

virtual T PopFront() { return 0; }

virtual T PopBack() { return 0; }

virtual int GetSize() { return 0; }

virtual T PeekFront() { return 0; }

virtual T PeekBack() { return 0; }

virtual void GetFirst() {}

};

template<typename T>

class DequeTemp : Base<T>

{

T\* arr;

int count;

int end = 0, size = 0;

public:

//инициализация

DequeTemp(int count)

: count(count)

{

this->count = count;

arr = new T[count];

}

//копирование

DequeTemp(const DequeTemp& other)

{

arr = new T[other.count];

count = other.count;

end = other.end;

size = other.size;

for (int i = 0; i < count; ++i)

{

arr[i] = other.arr[i];

}

}

//перемещение

DequeTemp(DequeTemp&& other)

{

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

//деструктор

~DequeTemp()

{

delete[] arr;

}

//присваивание

DequeTemp& operator=(const DequeTemp& other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = new T[other.count];

count = other.count;

for (int i = 0; i < count; ++i)

{

arr[i] = other.arr[i];

}

}

//присваивание с перемещением

DequeTemp& operator=(DequeTemp&& other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

int GetSize()

{

return this->size;

}

//Вставка элемента

void PushFront(T element)

{

if (size <= count)

{

int\* tmp = new int[size];

for (int i = 0; i <= size; i++)

{

tmp[i] = arr[i];

}

for (int i = 0; i <= size; i++)

{

arr[i + 1] = tmp[i];

}

arr[0] = element;

size++;

end++;

}

}

void PushBack(T element)

{

if (size <= count)

{

end += 1;

arr[end] = element;

size++;

}

}

//Удаление элемента

T PopFront()

{

if (size <= count)

{

for (int i = 0; i < end; i++)

{

arr[i] = arr[i + 1];

}

arr[end] = 0;

end--;

size--;

return 0;

}

}

T PopBack()

{

if (size <= count)

{

arr[end] = 0;

end--;

size--;

return 0;

}

}

//Просмотр элемента

T PeekFront()

{

return arr[0];

}

T PeekBack()

{

return arr[end];

}

void GetFirst(T element)

{

arr[0] = element;

size++;

}

//Вывод

friend ostream& operator<<(ostream& stream, DequeTemp& a)

{

for (int i = 0; i < a.size; i++)

stream << a.arr[i] << " ";

return stream;

}

};

size = other.size;

for (int i = 0; i < count; ++i)

{

arr[i] = other.arr[i];

}

}

//перемещение

DequeTemp(DequeTemp&& other)

{

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

//деструктор

~DequeTemp()

{

delete[] arr;

}

//присваивание

DequeTemp& operator=(const DequeTemp& other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = new T[other.count];

count = other.count;

for (int i = 0; i < count; ++i)

{

arr[i] = other.arr[i];

}

}

//присваивание с перемещением

DequeTemp& operator=(DequeTemp&& other)

{

if (this == &other)

{

return \*this;

}

delete[] arr;

arr = other.arr;

count = other.count;

other.arr = nullptr;

}

int GetSize()

{

return this->size;

}

//Вставка элемента

void PushFront(T element)

{

if (size <= count)

{

int\* tmp = new int[size];

for (int i = 0; i <= size; i++)

{

tmp[i] = arr[i];

}

for (int i = 0; i <= size; i++)

{

arr[i + 1] = tmp[i];

}

arr[0] = element;

size++;

end++;

}

}

void PushBack(T element)

{

if (size <= count)

{

end += 1;

arr[end] = element;

size++;

}

}

//Удаление элемента

T PopFront()

{

if (size <= count)

{

for (int i = 0; i < end; i++)

{

arr[i] = arr[i + 1];

}

arr[end] = 0;

end--;

size--;

return 0;

}

}

T PopBack()

{

if (size <= count)

{

arr[end] = 0;

end--;

size--;

return 0;

}

}

//Просмотр элемента

T PeekFront()

{

return arr[0];

}

T PeekBack()

{

return arr[end];

}

void GetFirst(T element)

{

arr[0] = element;

size++;

}

//Вывод

friend ostream& operator<<(ostream& stream, DequeTemp& a)

{

for (int i = 0; i < a.size; i++)

stream << a.arr[i] << " ";

return stream;

}

};

for (int i = 0; i <= size; i++)

{

arr[i + 1] = tmp[i];

}

arr[0] = element;

size++;

end++;

}

}

void PushBack(T element)

{

if (size <= count)

{

end += 1;

arr[end] = element;

size++;

}

}

//Удаление элемента

T PopFront()

{

if (size <= count)

{

for (int i = 0; i < end; i++)

{

arr[i] = arr[i + 1];

}

arr[end] = 0;

end--;

size--;

return 0;

}

}

T PopBack()

{

if (size <= count)

{

arr[end] = 0;

end--;

size--;

return 0;

}

}

//Просмотр элемента

T PeekFront()

{

return arr[0];

}

T PeekBack()

{

return arr[end];

}

void GetFirst(T element)

{

arr[0] = element;

size++;

}

//Вывод

friend ostream& operator<<(ostream& stream, DequeTemp& a)

{

for (int i = 0; i < a.size; i++)

stream << a.arr[i] << " ";

return stream;

}

};

for (int i = 0; i < a.size; i++)

stream << a.arr[i] << " ";

return stream;

}

};

Листинг 1.3 – Исходный код файла task2.cpp.

#include <iostream>

#include "deque2.cpp"

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus");

DequeTemp<int>\* a = new DequeTemp<int>;

DequeTemp<int>\* aclass;

a->PushFront(1);

a->PushFront(2);

a->PushBack(3);

a->PushBack(4);

cout << \*a << endl;

cout << "First = " << a->PeekFront() << endl;

cout << "Back = " << a->PeekBack() << endl;

cout << "Size = " << a->GetSize() << endl;

a->PopFront();

cout << "PopFrontDeque: " << \*a << endl;

a->PopBack();

cout << "PopBackDeque: " << \*a << endl;

aclass = a;

cout << "New Aclass = " << \*aclass << endl;

system("pause");

}

Листинг 1.4 – Исходный код файла deque2.cpp.

#include <iostream>

using namespace std;

template<typename T>

class Base

{

public:

virtual void PushFront(T element) {}

virtual void PushBack(T element) {}

virtual void PopFront() { }

virtual void PopBack() { }

virtual int GetSize() { return 0; }

virtual T PeekFront() { return 0; }

virtual T PeekBack() { return 0; }

virtual void GetFirst() {}

};

template<typename T>

class DequeTemp : Base<T>

{

public:

DequeTemp()

{

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

}

DequeTemp(const DequeTemp& other)

{

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(temp->data\_);

temp = temp->pNext\_;

}

PushBack(temp->data\_);

}

DequeTemp(DequeTemp&& other)

{

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(other.deQueue());

temp = temp->pNext\_;

}

PushBack(temp->data\_);

}

~DequeTemp()

{

while (size\_)

PopFront();

}

int GetSize()

{

return size\_;

}

void PushFront(T element)

{

if (head\_ == NULL)

{

head\_ = new Node<T>(element);

tail\_ = head\_;

}

else

{

head\_->pPrev\_ = new Node<T>(element, head\_, NULL);

head\_ = head\_->pPrev\_;

}

size\_++;

}

void PushBack(T element)

{

if (head\_ == NULL)

{

head\_ = new Node<T>(element);

tail\_ = head\_;

}

else

{

tail\_->pNext\_ = new Node<T>(element, NULL, tail\_);

tail\_ = tail\_->pNext\_;

}

size\_++;

}

void PopFront()

{

Node<T>\* temp = head\_; // Запоминаем адрес головного элемента

head\_ = head\_->pNext\_; // Смещаем голову списка на следующий узел

head\_->pPrev\_ = NULL;

delete temp; // Удаляем предыдущий за головой узел

size\_--;

}

void PopBack()

{

Node<T>\* temp = tail\_; // Запоминаем адрес хвостового элемента

tail\_ = tail\_->pPrev\_; // Смещаем хвост списка на предыдущий узел

tail\_->pNext\_ = NULL;

delete temp; // Удаляем следующий за хвостом узел

size\_--;

}

T PeekFront()

{

return head\_->data\_;

}

T PeekBack()

{

return tail\_->data\_;

}

DequeTemp<T>& operator=(const DequeTemp<T>& other)

{

if (this == &other)

return \*this;

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(temp->data\_);

temp = temp->pNext\_;

}

PushBack(temp->data\_);

}

DequeTemp<T>& operator=(DequeTemp<T>&& other)

{

if (this == &other)

return \*this;

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(other.PopFront());

temp = temp->pNext\_;

}

PushBack(temp->data);

}

friend ostream& operator<<(ostream& out, DequeTemp& q)

{

Node<T>\* temp = q.head\_;

while (temp->pNext\_ != NULL)

{

int test = temp->data\_;

out << temp->data\_ << " ";

temp = temp->pNext\_;

}

out << temp->data\_ << " ";

return out;

}

private:

template<typename T>

class Node

{

public:

Node(T data, Node\* pNext = NULL, Node\* pPrev = NULL)

{

data\_ = data;

pNext\_ = pNext;

pPrev\_ = pPrev;

}

Node\* pNext\_;

Node\* pPrev\_;

T data\_;

};

Node<T>\* head\_;

Node<T>\* tail\_;

int size\_;

};

virtual void GetFirst() {}

};

template<typename T>

class DequeTemp : Base<T>

{

public:

DequeTemp()

{

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

}

DequeTemp(const DequeTemp& other)

{

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(temp->data\_);

temp = temp->pNext\_;

}

PushBack(temp->data\_);

}

DequeTemp(DequeTemp&& other)

{

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(other.deQueue());

temp = temp->pNext\_;

}

PushBack(temp->data\_);

}

~DequeTemp()

{

while (size\_)

PopFront();

}

int GetSize()

{

return size\_;

}

void PushFront(T element)

{

if (head\_ == NULL)

{

head\_ = new Node<T>(element);

tail\_ = head\_;

}

else

{

head\_->pPrev\_ = new Node<T>(element, head\_, NULL);

head\_ = head\_->pPrev\_;

}

size\_++;

}

void PushBack(T element)

{

if (head\_ == NULL)

{

head\_ = new Node<T>(element);

tail\_ = head\_;

}

else

{

tail\_->pNext\_ = new Node<T>(element, NULL, tail\_);

tail\_ = tail\_->pNext\_;

}

size\_++;

}

void PopFront()

{

Node<T>\* temp = head\_; // Запоминаем адрес головного элемента

head\_ = head\_->pNext\_; // Смещаем голову списка на следующий узел

head\_->pPrev\_ = NULL;

delete temp; // Удаляем предыдущий за головой узел

size\_--;

}

void PopBack()

{

Node<T>\* temp = tail\_; // Запоминаем адрес хвостового элемента

tail\_ = tail\_->pPrev\_; // Смещаем хвост списка на предыдущий узел

tail\_->pNext\_ = NULL;

delete temp; // Удаляем следующий за хвостом узел

size\_--;

}

T PeekFront()

{

return head\_->data\_;

}

T PeekBack()

{

return tail\_->data\_;

}

DequeTemp<T>& operator=(const DequeTemp<T>& other)

{

if (this == &other)

return \*this;

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(temp->data\_);

temp = temp->pNext\_;

}

PushBack(temp->data\_);

}

DequeTemp<T>& operator=(DequeTemp<T>&& other)

{

if (this == &other)

return \*this;

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(other.PopFront());

temp = temp->pNext\_;

}

PushBack(temp->data);

}

friend ostream& operator<<(ostream& out, DequeTemp& q)

{

Node<T>\* temp = q.head\_;

while (temp->pNext\_ != NULL)

{

int test = temp->data\_;

out << temp->data\_ << " ";

temp = temp->pNext\_;

}

out << temp->data\_ << " ";

return out;

}

private:

template<typename T>

class Node

{

public:

Node(T data, Node\* pNext = NULL, Node\* pPrev = NULL)

{

data\_ = data;

pNext\_ = pNext;

pPrev\_ = pPrev;

}

Node\* pNext\_;

Node\* pPrev\_;

T data\_;

};

Node<T>\* head\_;

Node<T>\* tail\_;

int size\_;

};

}

void PushBack(T element)

{

if (head\_ == NULL)

{

head\_ = new Node<T>(element);

tail\_ = head\_;

}

else

{

tail\_->pNext\_ = new Node<T>(element, NULL, tail\_);

tail\_ = tail\_->pNext\_;

}

size\_++;

}

void PopFront()

{

Node<T>\* temp = head\_; // Запоминаем адрес головного элемента

head\_ = head\_->pNext\_; // Смещаем голову списка на следующий узел

head\_->pPrev\_ = NULL;

delete temp; // Удаляем предыдущий за головой узел

size\_--;

}

void PopBack()

{

Node<T>\* temp = tail\_; // Запоминаем адрес хвостового элемента

tail\_ = tail\_->pPrev\_; // Смещаем хвост списка на предыдущий узел

tail\_->pNext\_ = NULL;

delete temp; // Удаляем следующий за хвостом узел

size\_--;

}

T PeekFront()

{

return head\_->data\_;

}

T PeekBack()

{

return tail\_->data\_;

}

DequeTemp<T>& operator=(const DequeTemp<T>& other)

{

if (this == &other)

return \*this;

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(temp->data\_);

temp = temp->pNext\_;

}

PushBack(temp->data\_);

}

DequeTemp<T>& operator=(DequeTemp<T>&& other)

{

if (this == &other)

return \*this;

head\_ = NULL;

tail\_ = NULL;

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(other.PopFront());

temp = temp->pNext\_;

}

PushBack(temp->data);

}

friend ostream& operator<<(ostream& out, DequeTemp& q)

{

Node<T>\* temp = q.head\_;

while (temp->pNext\_ != NULL)

{

int test = temp->data\_;

out << temp->data\_ << " ";

temp = temp->pNext\_;

}

out << temp->data\_ << " ";

return out;

}

private:

template<typename T>

class Node

{

public:

Node(T data, Node\* pNext = NULL, Node\* pPrev = NULL)

{

data\_ = data;

pNext\_ = pNext;

pPrev\_ = pPrev;

}

Node\* pNext\_;

Node\* pPrev\_;

T data\_;

};

Node<T>\* head\_;

Node<T>\* tail\_;

int size\_;

};

size\_ = 0;

Node<T>\* temp = new Node<T>();

while (temp->pNext\_ != other.tail\_)

{

PushBack(other.PopFront());

temp = temp->pNext\_;

}

PushBack(temp->data);

}

friend ostream& operator<<(ostream& out, DequeTemp& q)

{

Node<T>\* temp = q.head\_;

while (temp->pNext\_ != NULL)

{

int test = temp->data\_;

out << temp->data\_ << " ";

temp = temp->pNext\_;

}

out << temp->data\_ << " ";

return out;

}

private:

template<typename T>

class Node

{

public:

Node(T data, Node\* pNext = NULL, Node\* pPrev = NULL)

{

data\_ = data;

pNext\_ = pNext;

pPrev\_ = pPrev;

}

Node\* pNext\_;

Node\* pPrev\_;

T data\_;

};

Node<T>\* head\_;

Node<T>\* tail\_;

int size\_;

};

Листинг 1.5 – Исходный код файла main.h.

#pragma once

using namespace std;

template<typename T>

struct UCLNode

{

T data;

UCLNode\* ptrnext;

UCLNode\* ptrprev;

};

template<typename T>

class UnidirCyclicList

{

private:

UCLNode<T>\* first;

UCLNode<T>\* last;

public:

UnidirCyclicList()

{

first = nullptr;

last = nullptr;

}

~UnidirCyclicList()

{

while (first != last)

{

UCLNode<T>\* tmp = first;

first = first->ptrprev;

delete tmp;

}

delete last;

}

void AddBack(T dt)

{

UCLNode<T>\* newlink = new UCLNode<T>;

newlink->data = dt;

if (first)

{

newlink->ptrprev = first;

first->ptrnext = newlink;

}

else last = newlink;

newlink->ptrnext = last;

last->ptrprev = newlink;

first = newlink;

}

void AddFront(T dt)

{

UCLNode<T>\* newlink = new UCLNode<T>;

newlink->data = dt;

if (last)

{

newlink->ptrnext = last;

last->ptrprev = newlink;

}

else first = newlink;

newlink->ptrprev = first;

first->ptrnext = newlink;

last = newlink;

}

T RemoveFront()

{

T data = first->data;

DeleteLink(first);

return data;

}

T RemoveBack()

{

T data = last->data;

DeleteLink(last);

return data;

}

T ElementAt(int index)

{

if (index > Length()) return -1;

UCLNode<T>\* temp = first;

int i = 0;

do

{

if (i == index) return temp->data;

temp = temp->ptrprev;

i++;

} while (temp != first);

return -1;

}

void DeleteLink(UCLNode<T>\* temp)

{

if (first)

{

if (temp->ptrnext == temp)

{

first = nullptr;

last = nullptr;

}

else

{

if (temp == last) last = last->ptrnext;

if (temp == first) first = first->ptrprev;

(temp->ptrnext)->ptrprev = temp->ptrprev;

(temp->ptrprev)->ptrnext = temp->ptrnext;

delete temp;

}

}

}

UnidirCyclicList operator=(UnidirCyclicList& l)

{

if (l.first)

{

UCLNode<T>\* temp = l.first;

do

{

AddFront(temp->data);

temp = temp->ptrprev;

} while (temp != l.first);

}

return \*this;

}

int Length()

{

if (first)

{

UCLNode<T>\* temp = first;

int i = 0;

do

{

temp = temp->ptrprev;

i++;

} while (temp != first);

return i;

}

else return 0;

}

void Clear()

{

while (first->ptrprev != last)

{

UCLNode<T>\* temp = first->ptrprev;

first = first->ptrprev;

delete temp;

}

last = nullptr;

first = nullptr;

}

};

Листинг 1.6 – Исходный код файла main.cpp.

#include "main.h"

1. Скриншоты

Task1

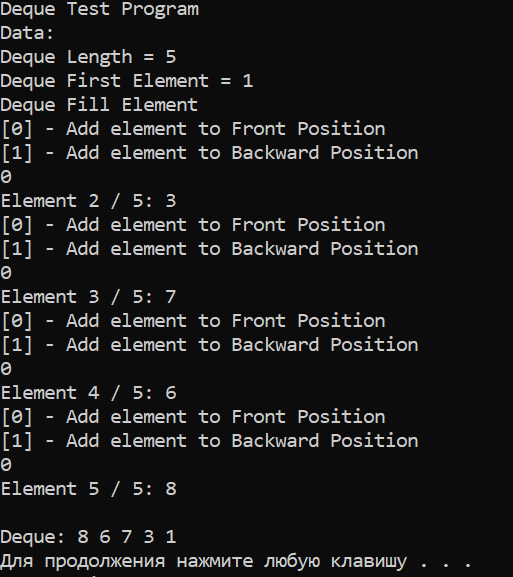


Рис. 1 – Добавление элементов

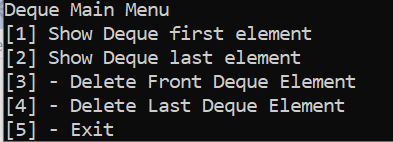


Рис. 2 – Возможные операции для данного дека



Рис. 4 – Реализация операции 1



Рис. 5 – Реализация операции 2

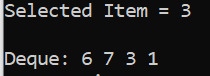


Рис. 6 – Реализация операции 3

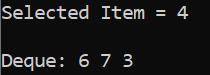


Рис. 7 – Реализация операции

**Task2**

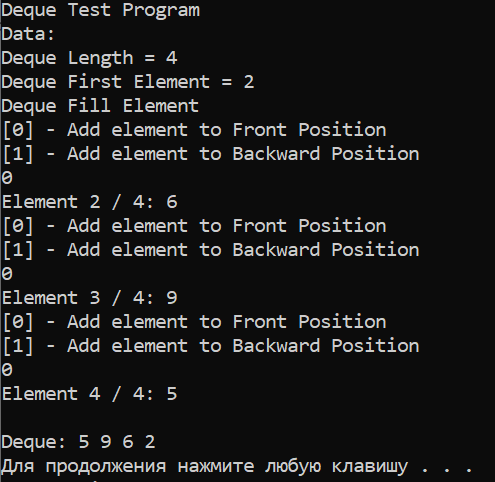
****

Рис. 1 – Добавление элементов

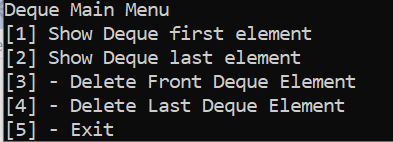


Рис. 2 – Возможные операции для данного дека



Рис. 4 – Реализация операции 1



Рис. 5 – Реализация операции 2

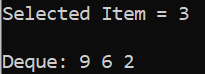


Рис. 6 – Реализация операции 3

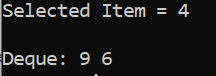


Рис. 7 – Реализация операции 4

1. Ссылка на репозиторий.

<https://github.com/Kseniia-ux/6-lab>